PROJECT REPORT

Celebrity Face Classification

Submitted by

Sanjeev Choubey Registration Number:- 11909115

Course Code INT246

Under the Guidance of

Dr Sagar Pande

School of Computer Science and Engineering

DECLARATION

We hereby declare that the project work entitled Celebrity face Recognition is an authentic record of our own work carried out as requirements of Project for the award of B.Tech degree in CSE from Lovely Professional University, Phagwara, under the guidance of (Name of Faculty Mentor), during August to November 2020. All the information furnished in this project report is based on our own intensive work and is genuine.

Sanjeev Choubey

11909115

[illegible]

CERTIFICATE

This is to certify that the declaration statement made by this group of students is correct to the best of my knowledge and belief. They have completed this Project under my guidance and supervision. The present work is the result of their original investigation, effort and study. No part of the work has ever been submitted for any other degree at any University. The Project is fit for the submission and partial fulfillment of the conditions for the award of B.Tech degree in CSE from Lovely Professional University, Phagwara

Signature and Name of the Mentor

Dr. Sagar Pande

Professor Lovely Professional University

School of Computer Science and Engineering,
Lovely Professional University,
Phagwara, Punjab.

Acknowledgment

Foremost, I would like to express my sincere gratitude to my mentor and advisor Prof, Sagar Pande for the continuous support of my project , for his patience, motivation, enthusiasm, and immense knowledge . His guidance helped me in all the time of project. I could not have imagined having a better advisor and mentor for my project.

Sanjeev Choubey

TABLE OF CONTENTS

Introduction

What is face recognition?

Face recognition is the process of taking a face in an image and actually *identifying* who the face belongs to. Face recognition is thus a form of **person identification.**

Early face recognition systems relied on an early version of facial landmarks extracted from images, such as the relative position and size of the eyes, nose, cheekbone, and jaw. However, these systems were often highly subjective and prone to error since these quantifications of the face were manually extracted by the computer scientists and administrators running the face recognition software.

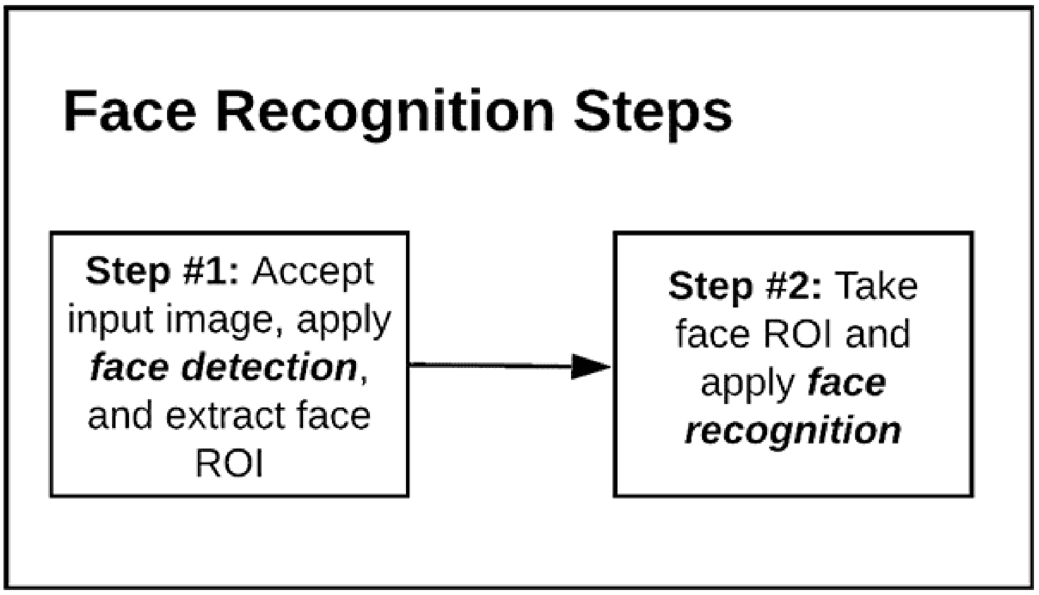
As machine learning algorithms became more powerful and the computer vision field matured, face recognition systems started to utilize feature extraction and classification models to identify faces in images.

Not only are these systems non-subjective, but they are also automatic — no hand labelling of the face is required. We simply extract features from the faces, train our classifier, and then use it to identify subsequent faces.

Most recently, we've started to utilize deep learning algorithms for face recognition. State-of-the-art face recognition models such as FaceNet and OpenFace rely on a specialized deep neural network architecture called **siamese networks**.

These neural networks are capable of obtaining face recognition accuracy that was once thought impossible (and they can achieve this accuracy with surprisingly little data).

How is face *recognition* different from face *detection*?


Face Recognition Steps
Step #1: Accept input image, apply ***face detection***, and extract face ROI
Step #2: Take face ROI and apply ***face recognition***

Face detection and face recognition are ***distinctly different*** algorithms — ***face detection*** will tell you ***where*** in a given image/frame a face is (but not ***who*** the face belongs to) while ***face recognition*** actually ***identifies*** the detected face.

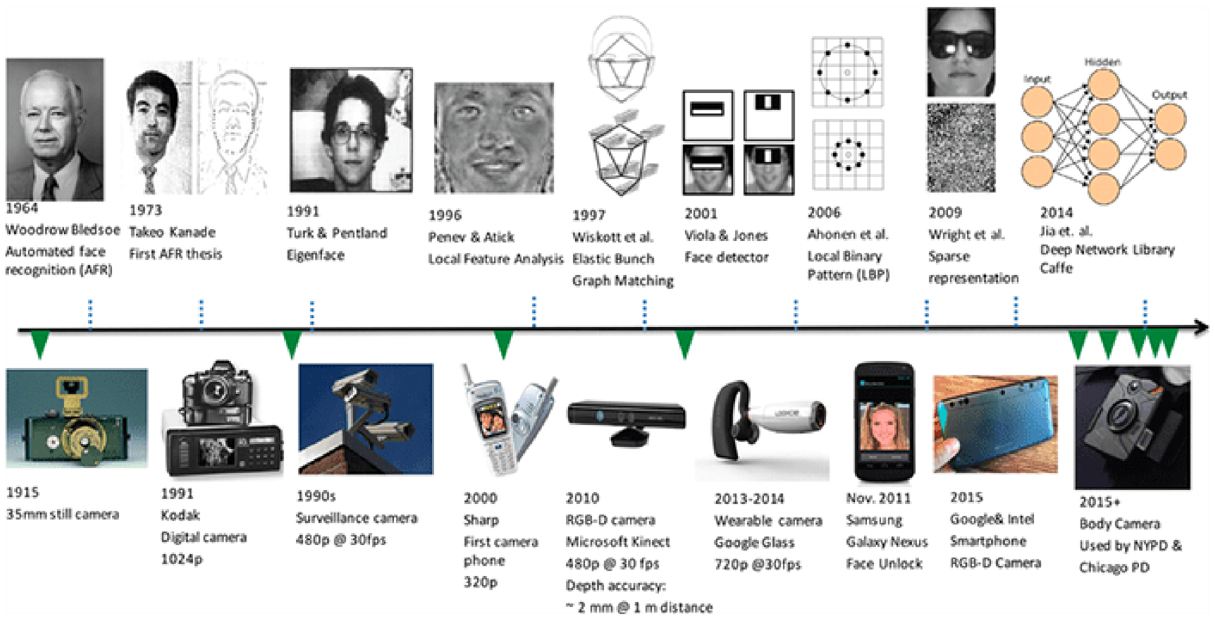
Unlike face detection, which is the process of simply detecting the presence of a face in an image or video stream, **face recognition** takes the faces detected from the localization phase and attempts to identify whom the face belongs to. Face recognition can thus be thought of as a method of person identification, which we use heavily in security and surveillance systems.

Since face recognition, by definition, requires face detection, we can think of face recognition as a two-phase process.

- **Phase 1:** Detect the presence of faces in an image or video stream using methods such as Haar cascades, HOG + Linear SVM, deep learning, or any other algorithm that can localize faces.

- **Phase 2:** Take each of the faces detected during the localization phase and identify each of them — this is where we actually assign a name to a face.

A brief history of face recognition


1964 Woodrow Bledsoe Automated face recognition (AFR)
1973 Takeo Kanade First AFR thesis
1991 Turk & Pentland Eigenface
1996 Penev & Atick Local Feature Analysis
1997 Wiskott et al. Elastic Bunch Graph Matching
2001 Viola & Jones Face detector
2006 Ahonen et al. Local Binary Pattern (LBP)
2009 Wright et al. Sparse representation
Input
Hidden
Output
2014 Jia et. al. Deep Network Library Caffe
1915 35mm still camera
1991 Kodak Digital camera 1024p
1990s Surveillance camera 480p @ 30fps
2000 Sharp First camera phone 320p
2010 RGB-D camera Microsoft Kinect 480p @ 30 fps Depth accuracy: ~ 2 mm @ 1 m distance
2013-2014 Wearable camera Google Glass 720p @30fps
Nov. 2011 Samsung Galaxy Nexus Face Unlock
2015 Google& Intel Smartphone RGB-D Camera
2015+ Body Camera Used by NYPD & Chicago PD

Face recognition may seem ubiquitous now (with it being implemented on most smartphones and major social media platforms), but prior to the 1970s, face recognition was often regarded as science fiction, sequestered to the movies and books set in ultra-future times. In short, face recognition was a fantasy, and whether or not it would become a reality was unclear.

This all changed in 1971 when Goldstein et al. published Identification of human faces. A crude first attempt at face identification, this method proposed 21 subjective facial features, such as hair color and lip thickness, to identify a face in a photograph.

The largest drawback of this approach was that the 21 measurements (besides being highly subjective) were *manually computed* — an obvious flaw in a computer science community that was rapidly approaching unsupervised computation and classification (at least in terms of human oversight).

Then, over a decade later, in 1987, Sirovich and Kirby published their seminal work, A Low-Dimensional Procedure for the Characterization of Human Faces which was later followed by Turk and Pentland in 1991 *with* Face Recognition Using Eigenfaces.

Deep learning is now responsible for unprecedented accuracy in face recognition. Specialized architectures called siamese networks are trained with a special type of data, called image triplets. We then compute, monitor, and attempt to minimize our triplet loss, thereby maximizing face recognition accuracy.

Pipeline Using OpenCV and SVM

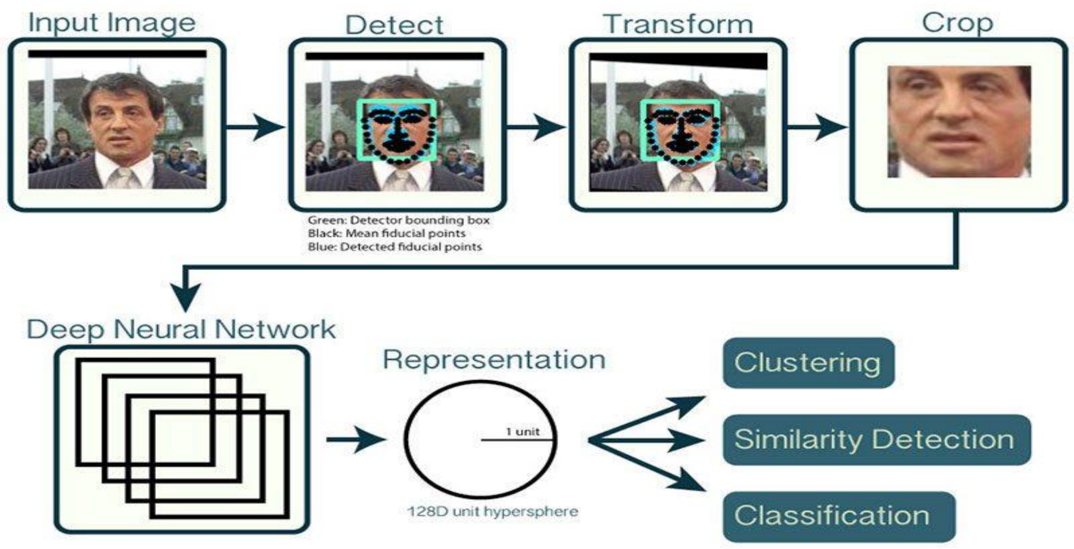


Image Pre Processing Using OpenCV

What is Image Processing?

Image processing involves performing some operations on an image, in order to get an enhanced image or to extract some useful information from it.
The field of image processing is upcoming and advancing rapidly. It enables object detection in images that has applications ranging from self driving cars to tumour detection in the field of medical science.

What are images?

What is a Image for computer?

What you see as an image is actually a 2D matrix for computer.

A digital image is stored as a combination of pixels. Each pixel further contains a different number of channels. If it a grayscale image, it has only one pixel, whereas a colored image contains three channels: **red, green**, and **blue.**

Each channel of each pixel has a value between **0 and 255**. With the combination of red, green and blue in different proportions we can create any colour.

Using OpenCV to Read Images in Python

In this tutorial, we will learn how to read images in Python using the OpenCV library.

OpenCV is an open-source computer vision and machine learning software library of programming functions mainly aimed at real-time computer vision.

Program to Read images

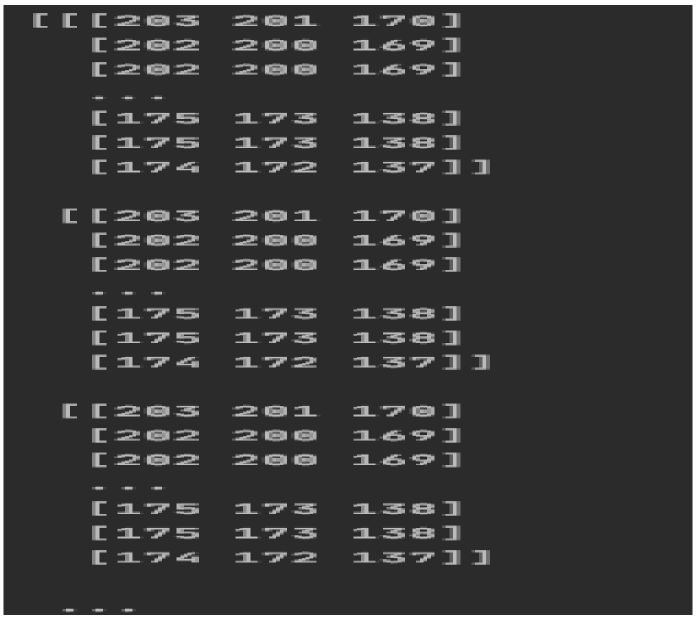
To read an image using OpenCV, use the following line of code.

```
img = cv2.imread('image_path')
```

Now the variable **img** will be a matrix of pixel values. We can print it and see the RGB values.

The image we are using for this example is:

```
print(img)
```



```
[[[203 201 170]
  [202 200 169]
  [202 200 169]
  ...
  [175 173 138]
  [175 173 138]
  [174 172 137]]

 [[203 201 170]
  [202 200 169]
  [202 200 169]
  ...
  [175 173 138]
  [175 173 138]
  [174 172 137]]

 [[203 201 170]
  [202 200 169]
  [202 200 169]
  ...
  [175 173 138]
  [175 173 138]
  [174 172 137]]

 ...
```

Display Images Using Python OpenCV

To show the image using OpenCV use the following line:

```
ccv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds.
The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If 0 is passed, it waits indefinitely for a keystroke.

Manipulating images using Python OpenCV

There are a lot of functionalities in OpenCV that allow you to manipulate an image. We will look at how to turn an image into grayscale.

A grayscale image means that each pixel will only have one channel with values between 0 to 255.

The code to do that is :

```
gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Save the manipulated image file

To save an image after manipulation use the following line of code:

```
cv2.imwrite('sample_grayscale.jpg',gray_image)
```

Here, the first argument is the name you want to give to the file, the second argument is the variable that contains the image you want to save. We are saving the grayscale image we created above.

Using OpenCV for rotating an Image

This is common that everyone knows that Python Open-CV is a module that will handle real-time applications related to computer vision. Open-CV works with image processing library **imutils** which deals with images. The **imutils.rotate()** function is used to rotate an image by an angle in Python.

Code Sample

```
import cv2 # importing cv

import imutils

# read an image as input using OpenCV

image = cv2.imread(r".\rotate.jpg")

Rotated_image = imutils.rotate(image, angle=45)

Rotated1_image = imutils.rotate(image, angle=90)

# display the image using OpenCV of

# angle 45
```

cv2.imshow("Rotated", Rotated_image)

# display the image using OpenCV of

# angle 90

cv2.imshow("Rotated", Rotated1_image)

# This is used for To Keep On Displaying

# The Image Until Any Key is Pressed

cv2.waitKey(0)

Cropping Using OpenCV

```
cropped_image = img[80:280, 150:330] # 
Slicing to crop the image

# Display 
the 
cropped 
image
cv2.imshow("cropped", 
cropped_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

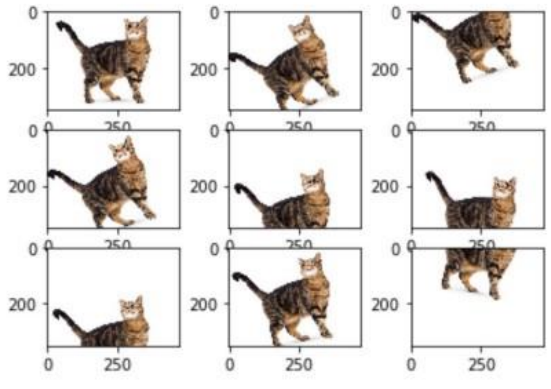
Image is represented by a 2d matrix so that we can slice it as it is.

Data Augmentation Techniques using OpenCV

Data augmentation is a strategy that enables practitioners to significantly increase the diversity of **data** available for training models, without actually collecting new **data**. **Data augmentation** techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks.

Data augmentation can be used to increased the accuracy and learning of the model because by using Data augmentation we can create several images of same image in different perspective like by flipping, rotating, sharpening and many more so that same image can be train in multiple ways and model can easily predict the image by so much learning .

Data Augmentation Techniques :



- Flipping: flipping the image vertically or horizontally.
- Rotation: rotates the image by a specified degree.
- Shearing: shifts one part of the image like a parallelogram.
- Cropping: object appear in different positions in different proportions in the image

There are more techniques for doing Data Augmentation by using various library , we can implement augmentation in a simple way by using ImageDataGenerator of keras but I have implemented using OpenCV.

Code Used for flipping images

```
def flip_img(image, flag):
#    horizontal flip
   if flag:
      return cv2.flip(img, 1)
#    vertical flip
   else:
      return cv2.flip(img, 0)
```

Feature Extraction Using Haar Cascades

Haar Cascade algorithm is one of the most powerful algorithms for the detection of objects specifically face detection in OpenCV proposed by Michael Jones and Paul Viola in their research paper called "Rapid Object Detection using a Boosted Cascade of Simple Features" and this algorithm was proposed in the year 2001which uses a function called cascade function for the detection of objects in the image and a lot of negative images and positive images are used to train this cascade function and this cascade function returns the image with rectangles drawn around the faces in the image as the output.

How to work haar cascade algorithm in OpenCV?

The haar cascade algorithm makes use of a kind of filter to perform feature extraction from the given image. These filters inspect only one portion of the image at a time. Then the intensity of the pixels in the white portion and in the black portion is added. The result of subtraction of these two summations is the feature extracted value.

There are three kinds of haar-like features extracted using the haar cascade algorithm, they are edge features, line features, and center-surround features. In order to detect the faces in the image, we load the pre-trained XML classifier file.

Then we make use of the detectMultiScale() function to detect the faces in the image. The positions of the detected faces are returned using the detectMultiScale() function. Then a region of Interest is drawn around the detected faces in the image. The detectMultiScale() function returns the image with rectangles drawn around the faces in the image as the output.

Example

OpenCV program in python to detect a face in the given image by using pre-trained haar cascade XML classifier and implementing haar cascade algorithm to display the image rectangles drawn around the faces in the image as the output on the screen:

```
import cv2

cascade_object= cv2.CascadeClassifier(‘Path’)

imageread = cv2.imread(‘Path')

imagegray = cv2.cvtColor(imageread, cv2.COLOR_BGR2GRAY)

face_positions = cascade_object.detectMultiScale(imagegray,
1.1, 4)


for (x, y, w, h) in face_positions:

cv2.rectangle(imageread, (x, y), (x+w, y+h), (0, 255, 0), 2)

cv2.imshow('Resulting_image', imageread)

cv2.waitKey()
```

Wavelet transform on images

The wavelet transform is similar to the Fourier transform (or much more to the windowed Fourier transform) with a completely different merit function. The main difference is this: ϖ Fourier transform decomposes the signal into sines and cosines, i.e. the functions localized in Fourier space; in contrary the wavelet transform uses functions that are localized in both the real and Fourier space.

As it is seen, the Wavelet transform is in fact an infinite set of various transforms, depending on the merit function used for its computation. This is the main reason, why we can hear the term “wavelet transform” in very different situations and applications. There are also many ways how to sort the types of the wavelet transforms. Here we show only the division based on the wavelet orthogonality. We can use orthogonal wavelets for discrete wavelet transform development and non-orthogonal wavelets for continuous wavelet transform development. These two transforms have the following properties:

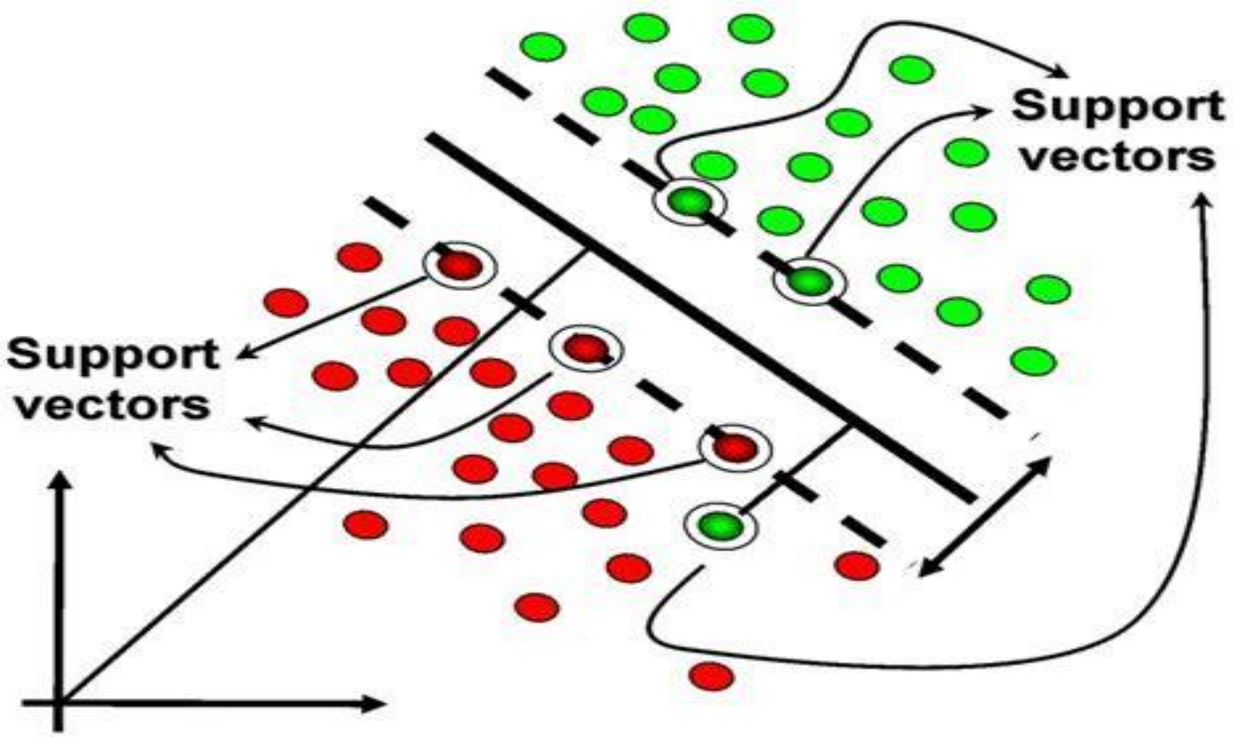
1. The discrete wavelet transform returns a data vector of the same length as the input is. Usually, even in this vector many data are almost zero. This corresponds to the fact that it decomposes into a set of wavelets (functions) that are orthogonal to its translations and scaling. Therefore we decompose such a signal to a same or lower number of the wavelet coefficient spectrum as is the number of signal data points. Such a wavelet spectrum is very good for signal processing and compression, for example, as we get no redundant information here.

2. The continuous wavelet transform in contrary returns an array one dimension larger than the input data. For a 1D data we obtain an image of the time-frequency plane. We can easily see the signal frequencies evolution during the duration of the signal and compare the spectrum with other signals spectra. As here is used the non-orthogonal set of wavelets, data are highly correlated, so big redundancy is seen here. This helps to see the results in a more humane form.

Support Vector Machine

Support vector machine in machine learning is defined as a data science algorithm that belongs to the class of supervised learning that analyses the trends and characteristics of the data set and solves problems related to classification and regression. Support vector machine is based on the learning framework of VC theory (Vapnik-Chervonenkis theory) and each of the training data points is marked as one of the 2 categories and then iteratively builds a region that will separate the data points in the space into 2 groups such that the data points in the region is well separated across the boundary with the maximum width or gap.

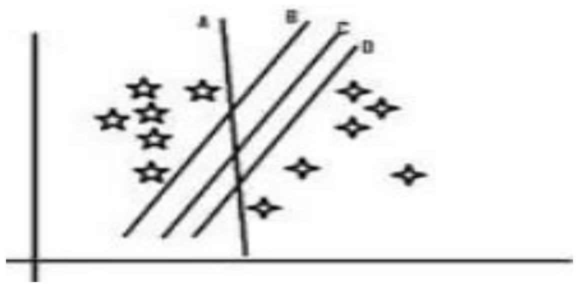
The setting up of different data points into respective one of the 2 categories is a non-probabilistic binary classifier. Now when a new example comes into the space, the corresponding group is predicted from the features and assigned the same group.



How Support Vector Machine Works in Machine Learning?

Support vector machine is able to generalize the characteristics that differentiate the training data that is provided to the algorithm. This is achieved by checking for a boundary that differentiates the two classes by the maximum margin. The boundary that separates the 2 classes is known as a hyperplane. Even if the name has a plane, if there are 2 features this hyperplane can be a line, in a 3-D it will be a plane, and so on. In the sample space, there is a possibility of zillions of hyperplanes but the objective in a support vector machine is to find that hyperplane through an iterative process that determines the hyperplane that has the maximum margin from all the corresponding training points and this will facilitate any new point coming in to fall into the accurate class on the basis of the features.

It is now time for us to know about the working of a support vector machine. There are 2 types of data and the approach in SVM towards the types of data is henceforth a bit different as well. For the first type of data, it needs to be linearly separable. Let us say that the data is spread across the sample space in the below way. For simplicity let us assume the data is a 2-D, and we will gradually move it to a 3-D as well in subsequent paragraphs.



In the above graph, we see that all the hyperplanes are accurately classifying the different groups we have in the data set. But the question is which of them does that with more cleanliness. Let us take each of them one by one. Line A does the job, but 2 points are very close to the line, and even a single movement might put the model into the confusion of which is correct and what is not. Now looking at line B, we see that the 5 pointer stars are very close to that line and hence still have the risk of getting misclassified even with a slight change. With line D, the 4-pointer stars are closer and hence stand the most chance of misclassification. With this, we are left with only line C that proves to be the best of the lot. The data points that are closest to the separable line are known as the support vectors and the distance of the support vectors from the separable line determines the best hyperplane

Advantages and Disadvantages of Support Vector Machine in Machine Learning

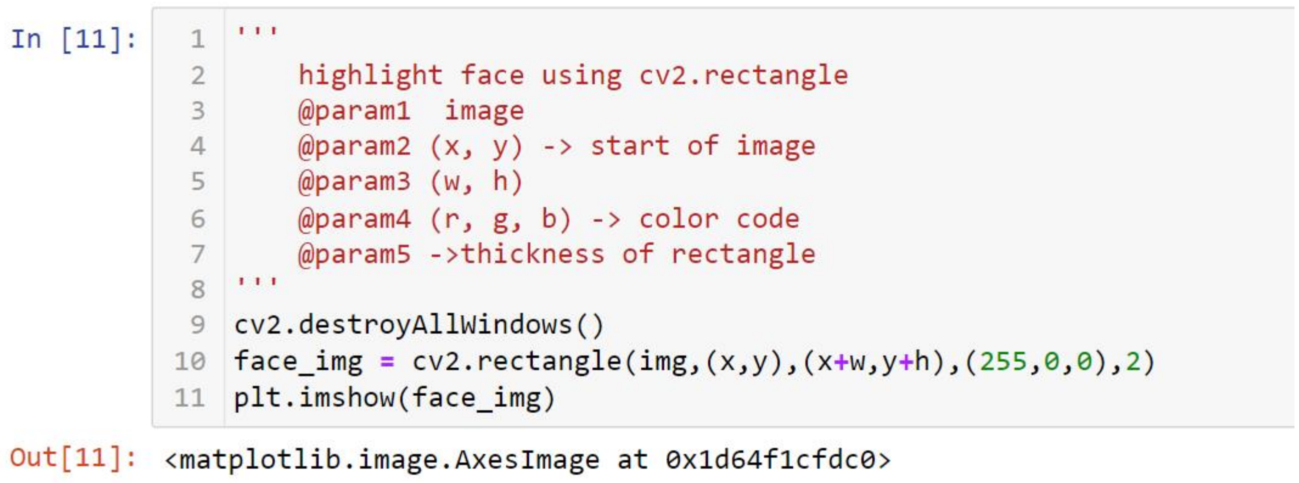
Advantages:

- Since only the support vectors are important to determine hyperplane, this algorithm has high stability.
- Outliers don’t influence the decision of hyperplane.
- Dataset need not have predefined assumptions.
- Again, since only support vectors are involved, this algorithm is memory efficient as well.

Disadvantages:

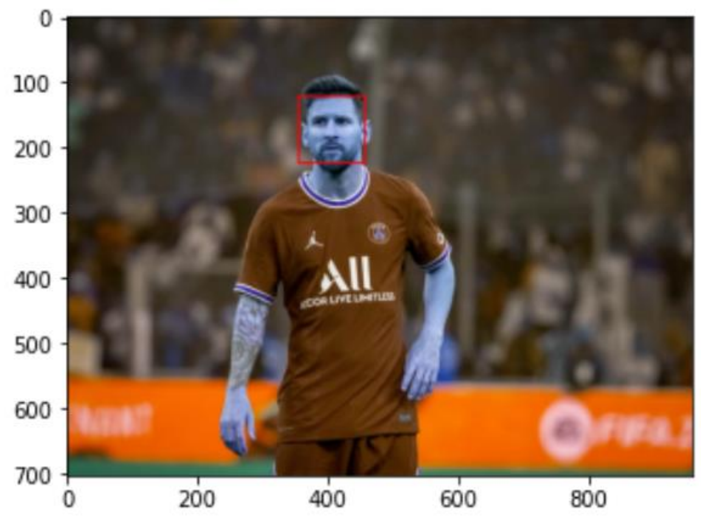
- The training time is higher when there is a large dataset.
- In case the target class is overlapping even at a higher dimension, SVM starts to perform poorly.
- This is a black box and doesn’t provide a probability estimate, which makes it even tougher to interpret during training on what to do next.

Project Screen Shots

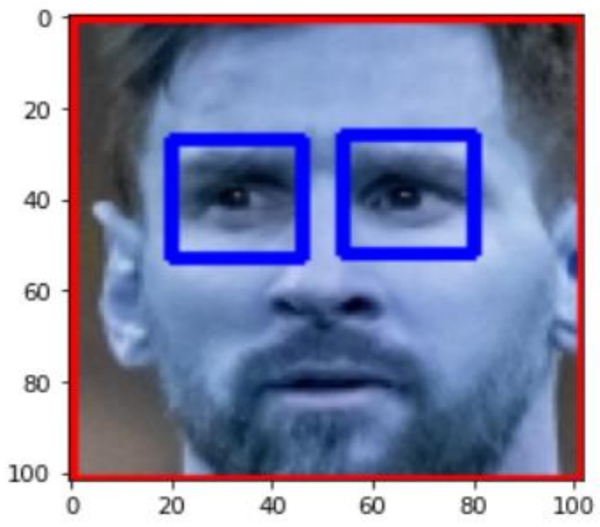


```
In [11]:
'''
    highlight face using cv2.rectangle
    @param1  image
    @param2 (x, y) -> start of image
    @param3 (w, h)
    @param4 (r, g, b) -> color code
    @param5 ->thickness of rectangle
'''
cv2.destroyAllWindows()
face_img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
plt.imshow(face_img)
```

```
Out[11]: <matplotlib.image.AxesImage at 0x1d64f1cfdc0>
```

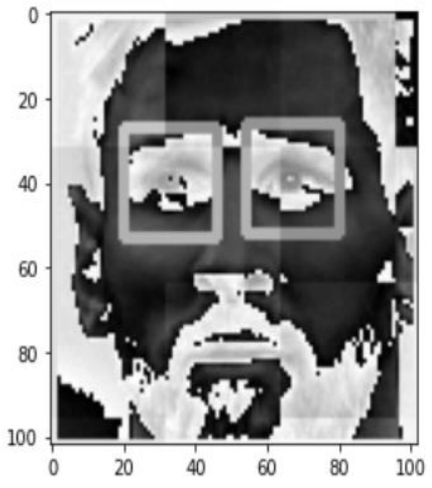


```
In [13]: '''
             show image inside red rectangle
         '''
         cv2.destroyAllWindows()
         plt.imshow(face_region_color, cmap='gray')

Out[13]: <matplotlib.image.AxesImage at 0x1d651c6c190>
```



Wavelet transformed image example:

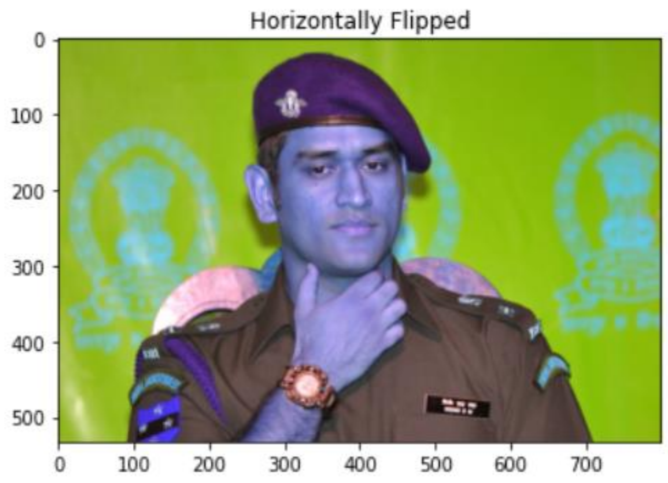
```
In [30]: waveLet = wavelet_transform(face_region_color, 'db1', 5)
         plt.imshow(waveLet, cmap='gray')

Out[30]: <matplotlib.image.AxesImage at 0x1d6543008b0>
```



```
'''
```

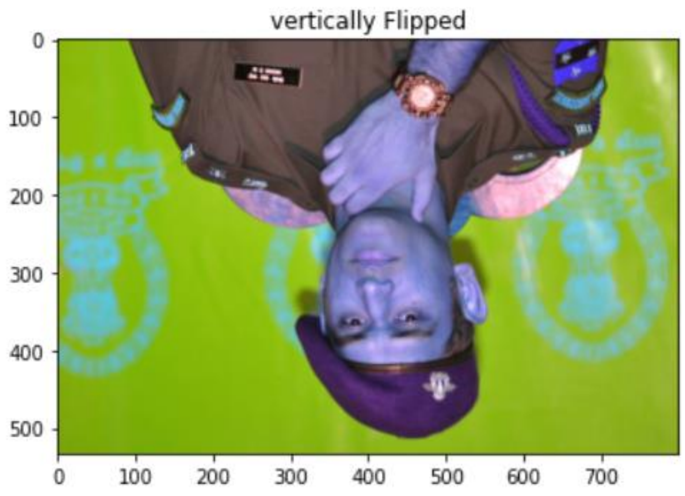
Horizontal flip Using OpenCV

```
In [9]:  original_img = cv2.imread('./Dataset/MsDhoni/dhoni1.jpg')
          flipped_img = flip_img(original_img, 1)
          plt.imshow(flipped_img)
          plt.title("Horizontally Flipped")

Out[9]:  Text(0.5, 1.0, 'Horizontally Flipped')
```



Vertical flip Using OpenCV

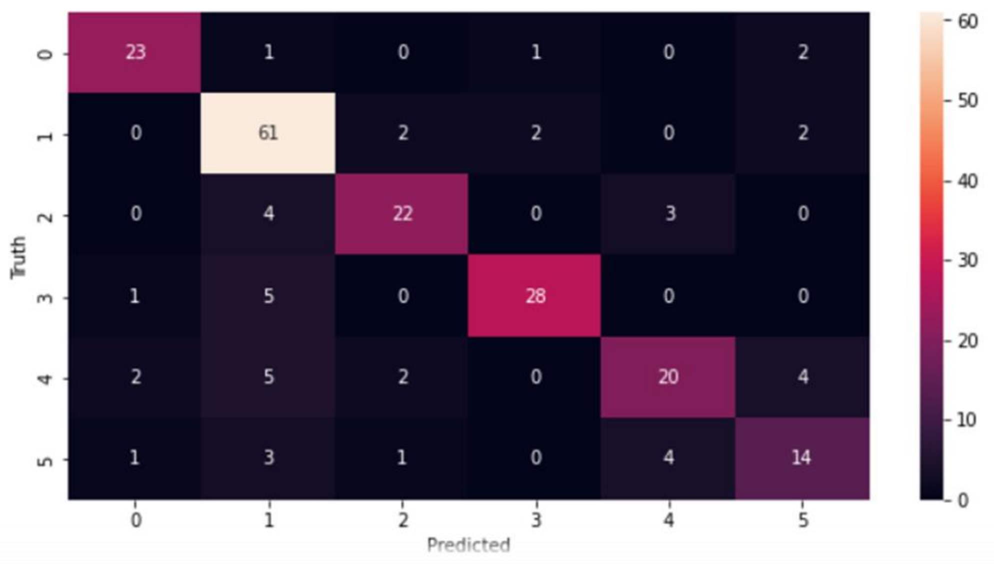
```
In [10]:  original_img = cv2.imread('./Dataset/MsDhoni/dhoni1.jpg')
           flipped_img = flip_img(original_img, 0)
           plt.imshow(flipped_img)
           plt.title("vertically Flipped")

Out[10]:  Text(0.5, 1.0, 'vertically Flipped')
```



Confusion Matrix Result

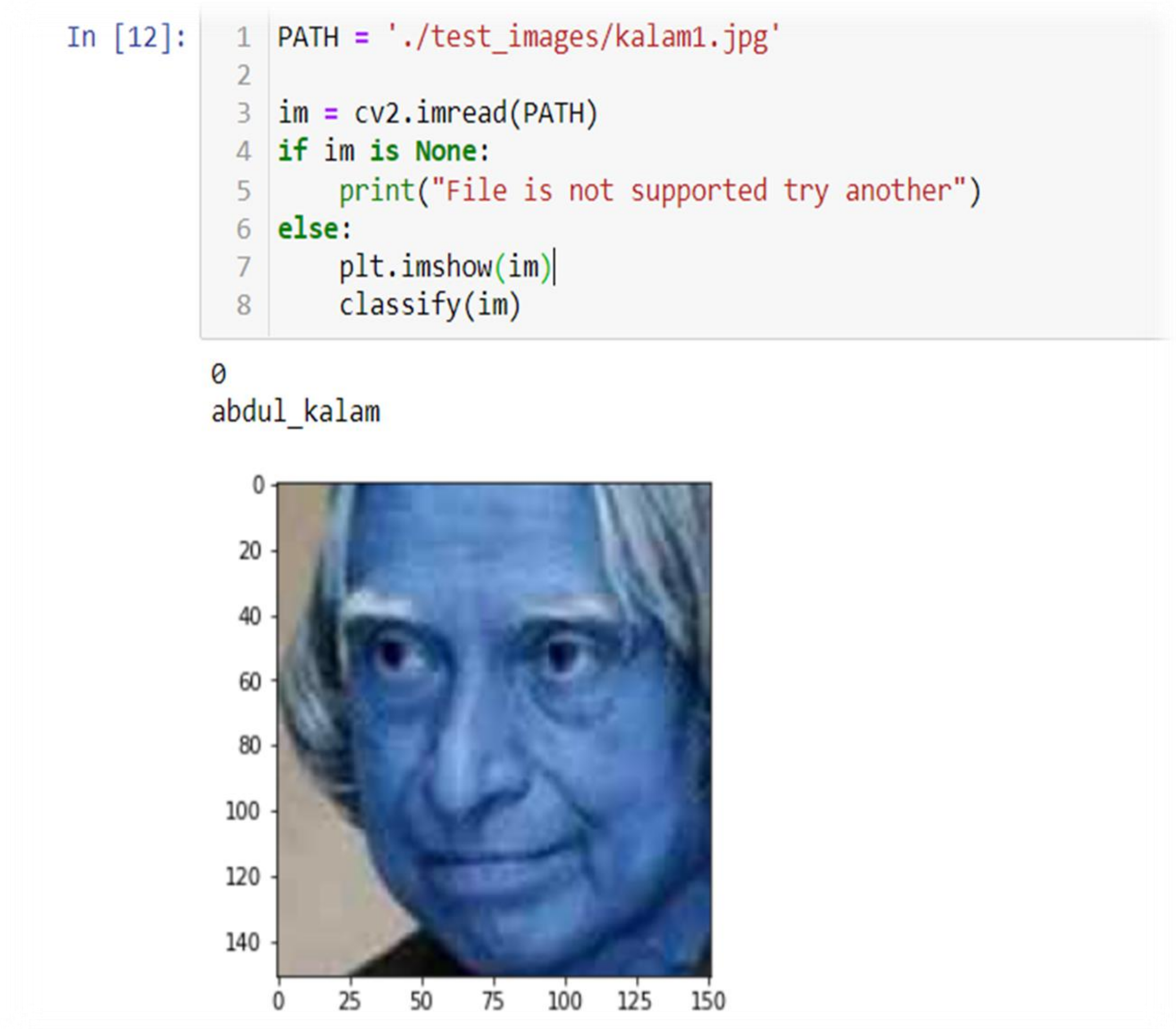
```
In [11]:
import seaborn as sns
plt.figure(figsize=(10, 5))
sns.heatmap(matrix_svm, annot = True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Out[11]: Text(69.0, 0.5, 'Truth')
```



Classification result

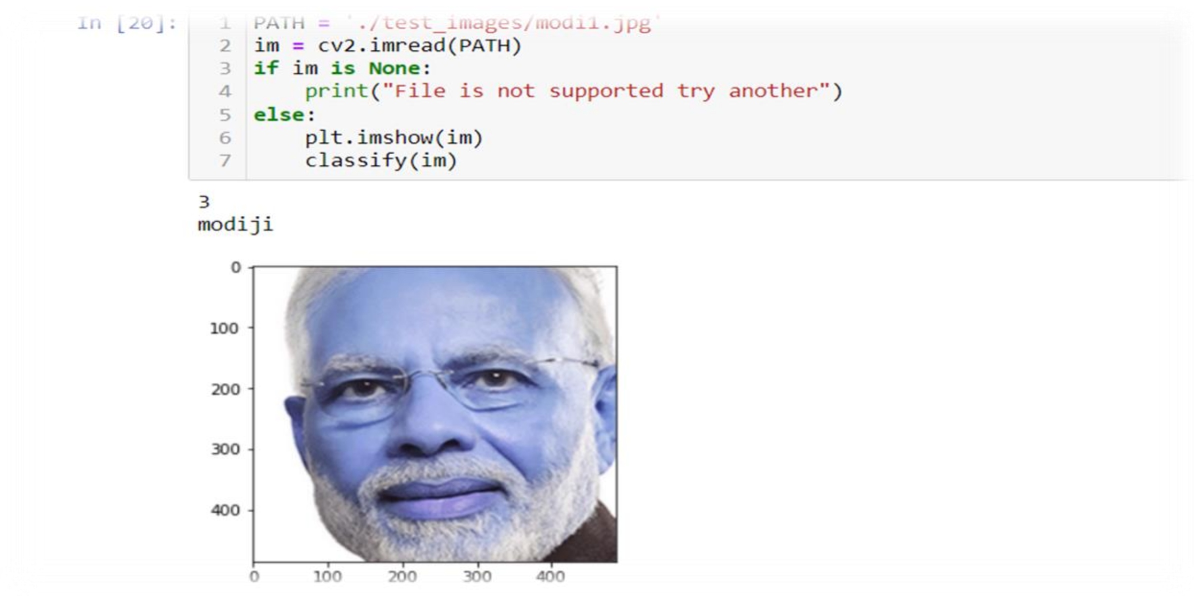
Dr APJ. Abdul Kalam



```
In [12]:
PATH = './test_images/kalam1.jpg'

im = cv2.imread(PATH)
if im is None:
    print("File is not supported try another")
else:
    plt.imshow(im)
    classify(im)
```

```
0
abdul_kalam
```

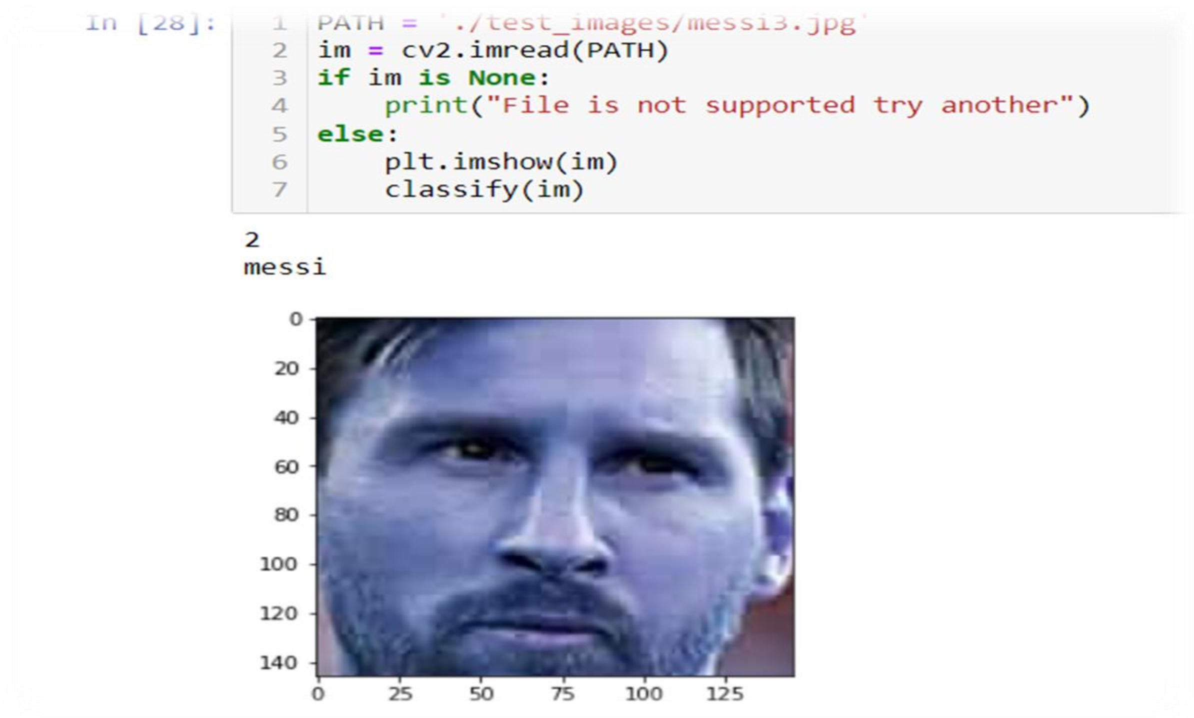
Narendra Modi



```
In [20]: PATH = './test_images/modi1.jpg'
         im = cv2.imread(PATH)
         if im is None:
             print("File is not supported try another")
         else:
             plt.imshow(im)
             classify(im)
```

```
3
modiji
```

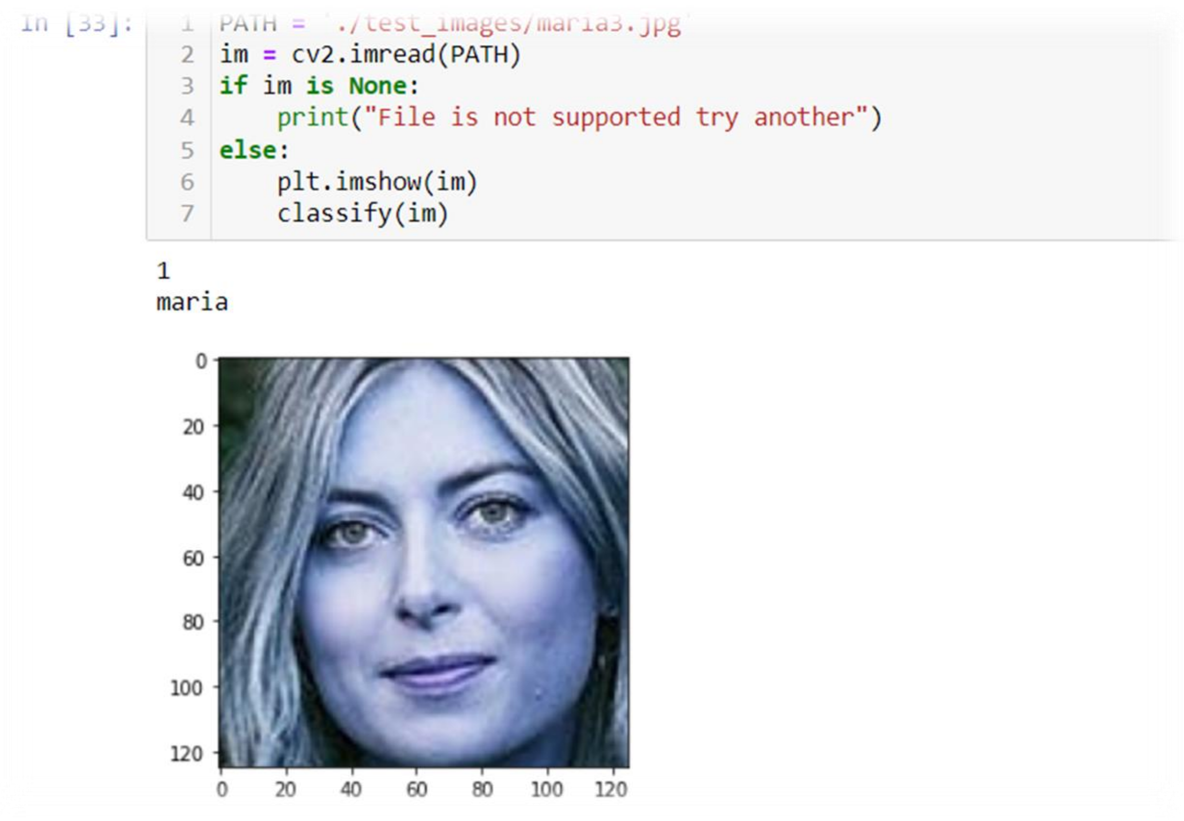
Lionel Messi



```
In [28]: PATH = './test_images/messi3.jpg'
         im = cv2.imread(PATH)
         if im is None:
             print("File is not supported try another")
         else:
             plt.imshow(im)
             classify(im)
```

```
2
messi
```

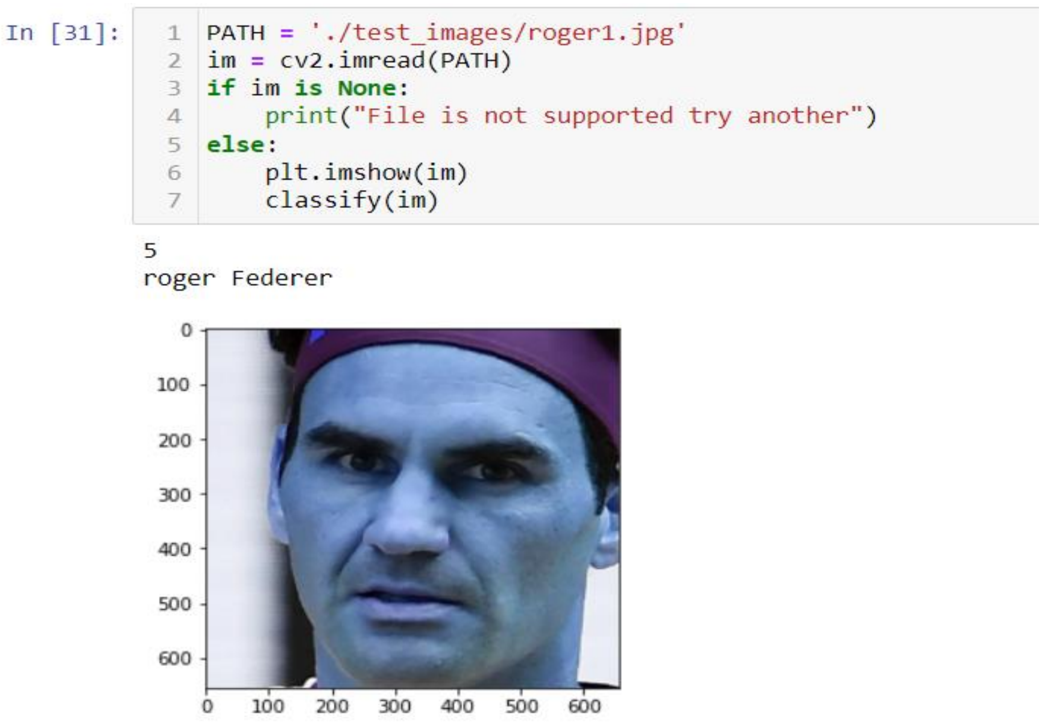
Maria Sharapova



```
In [33]:
PATH = './test_images/maria3.jpg'
im = cv2.imread(PATH)
if im is None:
    print("File is not supported try another")
else:
    plt.imshow(im)
    classify(im)
```

```
1
maria
```

Roger Federer



```
In [31]:
PATH = './test_images/roger1.jpg'
im = cv2.imread(PATH)
if im is None:
    print("File is not supported try another")
else:
    plt.imshow(im)
    classify(im)
```

```
5
roger Federer
```